基于改进慢启动算法的大文件快速传输 *

邓　彬[1]，成卫青[1,2]

(1. 南京邮电大学 计算机学院，南京 210003; 2. 东南大学 计算机网络和信息集成教育部重点实验室,南京 211189)

摘　要：针对传统 TCP 协议在当前网络环境下传输大文件性能较低的问题，对 TCP 传输协议中的慢启动算法部分进行了相应的研究与优化。根据标准慢启动算法存在的问题并结合高速网络以及大文件传输的性能特点，提出了一个具有网络状态感知能力的慢启动改进算法。改进算法主要优化了 TCP 拥塞窗口的增长策略，它实时地监测文件在传输过程中 TCP 报文段的往返时间(RTT)，并根据 RTT 的变化情况采用不同的窗口增长方式更新拥塞窗口；将改进算法部署在 Linux 网络模块中并分别在模拟网络环境和实际网络中进行测试。实验结果显示，改进算法能使发送窗口一直保持在一个较高的水平，实际数据传输速率和吞吐量均有了明显的提高。

关键词：传输协议；拥塞控制；大文件传输；性能评估

中图分类号：TP393..07　　doi: 10.19734/j.issn.1001-3695.2018.09.0645

Large file transfer based on improved slow start algorithm

Deng Bin[1], Cheng Weiqing[1, 2]

(1. *School of Computer*, *Nanjing University of Posts & Telecommunications*, *Nanjing* 210003, *China*; 2. *Key Laboratory of Computer Network & Information Integration of Ministry of Education*, *Southeast University*, *Nanjing* 211189, *China*)

Abstract: In view of the low performance of traditional TCP protocol in transmitting large files in the current network environment, this paper studied and optimized the slow start algorithm of TCP transport protocol. According to the problems of standard slow-start algorithm and the performance characteristics of high-speed network and large file transmission, an improved slow-start algorithm with network state awareness is proposed in this paper. The improved algorithm mainly optimizes the growth strategy of TCP congestion window, which monitors the round-trip time (RTT) of TCP packets in real-time during the transmission process, and updates the congestion window with different window growth methods according to the changes of RTT. It deployed the improved algorithm in the Linux network module and tested it in the simulated network environment and the actual network. Experimental results show that the improved algorithm can keep the sending window at a higher level, and the actual data transmission rate and throughput have been significantly improved.

Key words: transport protocol; congestion control; large file transfer; performance evaluation

0　引言

TCP 是当前互联网中普遍使用的传输层协议，它是一种面向连接、可靠并且有序的网络协议。据统计显示，当前互联网中 95%的数据流量均是通过 TCP 来传输的，占互联网总流量的 80%[1,2]，因此 TCP 传输协议的好坏对网络的整体性能有着很大的影响。网络拥塞是影响 TCP 传输效率的主要原因之一，随着社会信息化程度的不断提高，社会各个领域的大量数据都需要通过网络来传输，这使得网络传输效率和网络拥塞问题日益突出。1988 年，Van Jacobson 为了完善 TCP 的拥塞控制机制提出了 TCP Tahoe 算法，该算法在 TCP 中添加了“慢启动”“拥塞避免”以及“快速重传”，而后提出的 TCP Reno 算法又在此基础上增加了“快速恢复”机制[3]。经过多年的发展，TCP 已经从原始的 TCP Tahoe 发展到当前普遍使用的 TCP CUBIC[4]，拥塞控制算法也随之得到了进一步完善和改进，这大大提高了 TCP 的传输性能。然而，由于标准的 TCP 采用了 AIMD（加性增，乘性减）算法的窗口调整策略，使得数据传输在高速网络中产生了较大的时延抖动和带宽利用率不足等问题[5]，严重影响了网络的传输效率。慢启动是 TCP 在启动阶段用来探测未知网络可用带宽的关键算法，在 TCP 连接启动时期和分组发生超时重传的时候都会经历慢启动阶段，它通过增加发送窗口来不断地向网络中注入数据，以探测当前网络的带宽容量，在拥塞窗口未达到慢启动阈值的时候，窗口的大小在每接收到一个 ACK 时增加 1 个 MSS（最大报文段长度），这样在一个 RTT 时间内，拥塞窗口就会以指数倍的数量级扩大，然而这导致一个问题是在慢启动阶段后期窗口会由于增长过快，造成网络不必要的拥塞和分组丢失，从而影响了数据的传输。另外，慢启动阈值的盲目设定也不同程度地影响了 TCP 的传输性能，如果阈值 SSthresh 设置过小，会使得 TCP 连接提前进入拥塞避免阶段，无法有效地利用网络带宽资源，数据的传输速度也会随之降低。而如果阈值 SSthresh 设置过大，会导致拥塞窗口增长过大，大量的数据分组涌入网络，引起网络发生拥塞，过多的分组丢失和超时重传，严重影响了数据的传输效率，所以慢启动阈值的准确设定与否对 TCP 的传输性能也是非常关键的。研究表明[6]虽然大多数的 TCP 连接均为短连接，但少量的长连接却承载着网络中大部分的流量。此前，很多专家学者的研究热点也主要是集中在如何提高 TCP 短连接

收稿日期：2018-09-11；**修回日期：**2018-10-19　**基金项目：**计算机网络和信息集成教育部重点实验室资助项目（K93-9-2014-04B）；国家自然科学基金资助项目（61170322）

作者简介：邓彬（1992-），男，硕士研究生，主要研究方向为计算机网络；成卫青（1972-），女(通信作者)，教授，博士，主要研究方向为网络测量、模式识别(chengweiq@njupt.edu.cn).

的传输性能上，对 TCP 的长连接研究较少。本文的研究点主要集中在通过改善慢启动算法来提高 TCP 长连接的传输效率，从而加快大文件传输的速度。

1　慢启动算法存在的问题及相关研究

TCP 是通过调整拥塞窗口的大小来控制发送数据的速率的，因此，如何根据不同的网络状况来准确地探测网络的可用带宽进而相应地准确调整拥塞窗口是保持 TCP 连接始终处于高效率状态的关键所在。当前，已经有许多关于提高 TCP 慢启动性能的方法被提出，主要包括更加精细化的速率探测、基于带宽的评估、基于历史信息的预测以及通过路由器辅助的这四类，而这些方法研究的工作也主要是集中在初始拥塞窗口 initcwnd 大小的设定，慢启动阈值 ssthresh 的选取以及拥塞窗口 cwnd 的增长方式三个方面。

1) 初始拥塞窗口值的选取

由于传统的慢启动算法将拥塞窗口的初始值设为 1（其中 Linux 操作系统的网络模块默认将 TCP 初始拥塞窗口大小设为 3），这很显然限制了 TCP 连接在启动阶段拥塞窗口的增长速率，进而在一定程度上影响了 TCP 流的传输速率。因此有学者提出，通过将初始拥塞窗口提高到 3 可以明显提高 TCP 流的传输速率和性能[7]。另外，谷歌公司在 2010 年也发表了一篇论文，论文中提出在高延时高带宽的网络环境下通过将初始拥塞窗口增加到 10 可以大大提高 TCP 短流的传输效率并缩小网络的传输延迟[8]。虽然，通过单单地增加 initcwnd 值的大小可以有效提高 TCP 流的传输速率，但如果初始拥塞窗口值设置过大，不仅不会提高 TCP 的传输性能，而且还可能会因为网络拥塞导致过多的报文丢失进而使得 TCP 传输速率严重下降，所以在提高数据传输速率上，不能盲目地增大初始拥塞窗口，要充分地结合当前的网络环境和能力来选取一个最佳的值，这样不仅可以提高数据传输的速率，而且可以有效地提高带宽利用率。不管怎样，从谷歌的研究报告中可以清晰地知道初始拥塞窗口值的大小是影响 TCP 传输性能的一个重要因素。

2) 慢启动阈值的选取

选择合适的慢启动阈值不仅有助于拥塞窗口快速准确地收敛，还可以提高带宽的利用率以及减少数据包的丢失、重传。适当的慢启动阈值应该接近于 BDP（带宽时延积），盲目的设置可能会产生 TCP 传输过慢、带宽利用率低或者是因为注入过多数据包而造成严重的丢包等不同问题。受限慢启动 (limited slow start)[9] 通过引入一个新的慢启动阈值 max_SSthresh 来防止启动阶段产生过快的发送速率。无阈值启动(SSthreshless start)[2]则直接抛开了慢启动阈值的设定，通过动态的往返时延 RTT 来实时监控并利用瓶颈缓冲区的积压状态来优化拥塞窗口的增长行为，并自适应地调整数据发送速率以匹配当前的网络带宽。

3) 拥塞窗口的增长方式

在传统的慢启动阶段中，拥塞窗口是按指数方式增长的，这一方式的缺点是在窗口增长后期一个 RTT 的时间内就会向网络中注入大量的数据包，很容易造成数据包丢失甚至是引起网络阻塞[10]。Smooth start[11]假设默认的慢启动阈值大于 BDP，在窗口接近阈值的时候，通过减缓拥塞窗口的增长速度来避免过多的数据涌入网络。TCP Vegas[12]是一种基于时延反馈的协议，它根据 RTT 的时延变化来动态地检测网络的拥塞状况，并且在慢启动阶段采取了更加谨慎地做法，每隔一个 RTT 时间才会以指数倍的速度增加拥塞窗口，并且窗口固定时比较发送速率和期望速率，当实际发送速率小于期望速率一定值（设定的阈值 γ）的时候就会从慢启动模式转为拥塞避免模式，但其缺点是在高带宽高时延网络中带宽利用率比较低，传输效率不高。SS-IM 算法[15]则是通过将慢启动分为多个不同阶段并且每个阶段的拥塞窗口增长方式也不同，以此来优化慢启动。

2　基于网络状态感知的慢启动算法

针对前面介绍的关于慢启动算法存在的问题，本文提出一种具有网络状态感知能力的慢启动算法，简称改进的慢启动算法。其主要基于以下两个方面进行改进：a）通过增大初始拥塞窗口的大小来提高 TCP 链接前期的带宽利用率；b）通过动态测量 RTT 的变化来实时地调整拥塞窗口的增长方式。改进的慢启动算法试图准确实时地预测网络的拥塞状况并根据可用带宽的大小来相应地调整拥塞窗口，在尽量避免发生网络拥塞以及数据分组丢失重传的前提下，使得发送窗口一直保持在一个较高的水平，以此来提高网络的吞吐量和带宽的利用率。传统的拥塞控制算法一般都是基于有线的网络环境来设计的[12]，为了使算法更有通用性，本次改进的慢启动算法考虑到了无线网络链路质量较差的情况。算法的改进采用了稳中求进的策略，在尽量避免发生拥塞的前提下提高网络吞吐量。改进的慢启动算法在 TCP 连接刚建立时或者是在网络环境较好的情况下，和传统的慢启动算法类似，拥塞窗口是以指数的方式增加，增长速度较快，但是当网络环境开始变差，或者临近发生拥塞的情况下，拥塞窗口的增长速度会相应地减缓，以此来平滑地接近慢启动阈值，防止过多数据分组注入网络。另外，在谨慎增加拥塞窗口的同时也考虑到了通过增加初始拥塞窗口来提高数据的传输速度和带宽利用率，据 2009 年的一份研究报告[14]显示，全球平均的网络连接带宽达到了 1.7 Mbps,而且超过 50%的客户端带宽超过了 2 Mbps，所以本算法为了使窗口在起步时能以较高的起点和较快的速度增长，就结合了谷歌公司提出的对初始拥塞窗口大小设置的建议[8]，改进的算法增加了初始拥塞窗口的大小，结合实验取最优将初始拥塞窗口值设为 10。改进算法引入了一个根据实时的网络状况来动态调整拥塞窗口的增长因子 *diff*，即

$$diff = \frac{avgRTT - \min RTT}{\min RTT} \tag{1}$$

其中：由于 $avgRTT \geq \min RTT$ ，所以可得到 diff≧0。

Algorithm 1　Slow start algorithm

参数定义：*cwnd* 表示以 MSS（最大报文段长度）为单位的当前拥塞窗口；*n* 表示简单计数器；*curRTT* 表示当前的往返时延 RTT；***hRTT*** 表示算法采集的前 n-1 个往返时延值的简单算术平均值；***avgRTT***:表示 hRTT 和当前 RTT 的简单算术平均值；α 表示阈值因子（常量）；β 表示平衡因子（常量）；minRTT 表示所采集到的最小往返时延，初始值取为∞；*diff* 表示基于采集的 RTT 历史数据和最小 RTT 计算得来的值；*ssthresh* 表示慢启动阈值（取默认值）。

伪代码如下：

```
for every ACK
  if (cwnd<ssthresh) then //慢启动阶段
    n++;
    if (n==1) hRTT = curRTT;
    minRTT = min(minRTT,curRTT);
    avgRTT = (hRTT+curRTT)/2;
    diff = (avgRTT-minRTT)/minRTT;
    hRTT = ( hRTT*(n-1)+curRTT)/n;
```

```
        if cwnd < ssthresh/2 then
            cwnd+=1;
        else if cwnd >= ssthresh/2 && diff <=α then
            cwnd+=1/(diff+β);
        else if cwnd >= ssthresh/2 &&diff >α then
            cwnd+=1/(2*(diff+β));
    else
        congestion avoidance //拥塞避免阶段
end
```

TCP 每收到一个 ACK 置位的报文后且 *cwnd* 小于 *ssthresh* 时执行慢启动算法。Algorithm1 描述了改进的慢启动算法的具体细节。*diff* 是基于采集的历史数据计算得来的值，它是用来评估当前网络拥塞状况的关键因子，根据 *diff* 值的变化情况来预测当前网络的拥塞情况，然后 *diff* 会和 α 阈值因子进行比较并根据比较的结果来采取相应的窗口增长方式。基于 RTT 的平均值 *avgRTT* 和 *minRTT* 的比较更新 *cwnd* 可以很好地应对网络瞬间的不稳定情况，避免拥塞窗口大幅度地增加或减少。依靠 *minRTT*，*curRTT*，*avgRTT* 的取值并通过计算得来的 *diff* 将改进的慢启动算法分成了以下三种情形：

a) 当拥塞窗口 *cwnd* 小于慢启动阈值 *ssthresh* 的 1/2 时，表示当前的网络环境较为通畅，发生网络拥塞或数据包丢失的概率水平较低，*minRTT* 和 *avgRTT* 之间的差值较小，故采用同传统慢启动算法相一致的窗口增长策略，以指数的方式增加拥塞窗口。

b)在拥塞窗口 *cwnd* 大于 ssthresh/2 的情况下，改进算法分为了以下两种情况，即阈值因子 α 小于等于 3 和阈值因子大于 3。其中，阈值因子 α 的选取是根据实验证明得来，实验分别选取了 α=1、α=3、α=5，并测得相同网络环境下拥塞窗口的增长趋势。结合实验结果（图 3）可知，当 α 取值为 3 时，网络带宽利用率和吞吐量均相对较高。*diff* 小于等于阈值因子 α 时，表示当前网络环境相对良好，但发生网络拥塞的可能性有所增加，*minRTT* 和 *avgRTT* 之间的差值也有所增大，故本算法在此阶段的窗口增长机制放弃了以指数的方式增长，而是采用了一种相对保守且灵活的增长方式，即：$cwnd += \frac{1}{diff+\beta}$。其中 β 是平衡因子（β≧2）经实验证明并结合理论分析 β 取值为 2 时算法性能相对较高，另外根据式(1)可知 $diff \geq 0$，所以可得 $2 \leq diff+\beta \leq 5$，每个 RTT 拥塞窗口的增量范围为 $\frac{cwnd}{5} \leq \Delta cwnd \leq \frac{cwnd}{2}$，*cwnd* 增长的快慢完全取决于 *diff* 的大小。*diff* 值越大，拥塞窗口的增长速度就会相对减慢；*diff* 值越小，拥塞窗口的增长速度就会相对加快；

c)在拥塞窗口 *cwnd* 大于 *ssthresh*/2 的情况下，如果 *diff* 大于阈值因子 α，则表示当前网络环境开始变差，发生网络拥塞的概率水平进一步加大，丢包事件也进一步增多，往返时延 RTT 明显增大，所以在这种网络环境下，拥塞窗口的增长方式采用 $cwnd += \frac{1}{2*(diff+\beta)}$，即比在 $diff \leq 3$ 的情况下每个 RTT 拥塞窗口增量进一步减小，拥塞窗口的增量范围变为 $\Delta cwnd < \frac{cwnd}{10}$，这样窗口的增长就会更加谨慎，产生丢包重传的概率和窗口抖动的幅度也会相对地减小。

3 改进慢启动算法的实现和评估

Linux 是一款功能强大的网络操作系统，因此它被普遍用来当做客户端和服务器端的操作系统，而且 Linux 中已经实现了多种 TCP 拥塞控制算法，包括 Cubic[4]、Vegas[12]以及 Westwood[15]等。所以本次实验是通过修改 Linux 内核源码中的相应网络模块来测试和验证改进算法的各方面性能的，实验采用的 Linux 内核版本是 2.6.32，为了更加贴近实际，本文分别在模拟的网络环境下和实际的网络环境下评估改进算法。

本文利用 Linux 自带的 TC 工具来模拟网络环境[17~19]，通过设置不同的网络带宽、传输时延、丢包率、重复数据等性能参数，来模拟高速网络的网络环境，模拟网络的拓扑结构采用了典型的哑铃型拓扑结构，如图 1 所示。

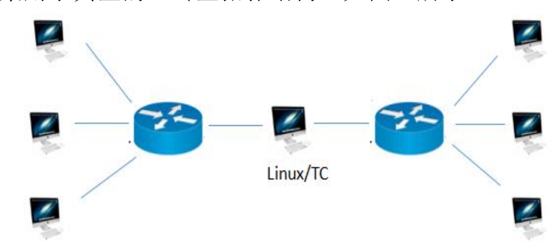


图 1 哑铃型的拓扑结构

Fig. 1 Dumbbell topology

3.1 模拟网络实验部分

在网络带宽设为 10 Mbps，传输时延为 100 ms，丢包率为 1%的网络环境下，分别使用改进的慢启动算法和传统的慢启动算法进行文件传输，文件的传输使用了 FTP 应用，接受方（即 FTP 应用服务端）初始通告窗口为 28 960 Byte，同时在文件传输的过程中采集拥塞窗口值，拥塞窗口变化如图 2 和 3 所示。

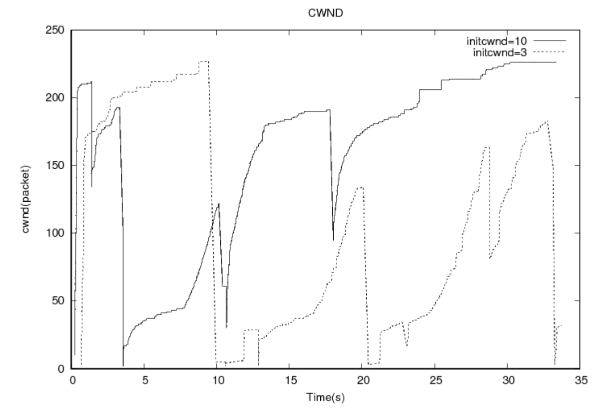


图 2 传统算法拥塞窗口变化趋势

Fig. 2 Changing trend of congestion window for traditional algorithm

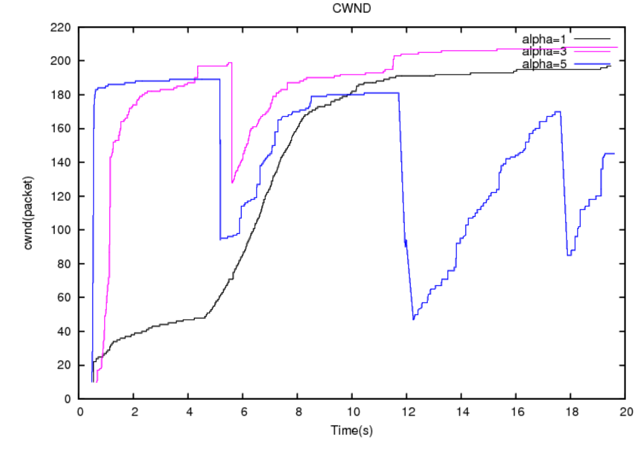


图 3 改进算法拥塞窗口变化趋势

Fig. 3 Changing trend of congestion window for improved algorithm

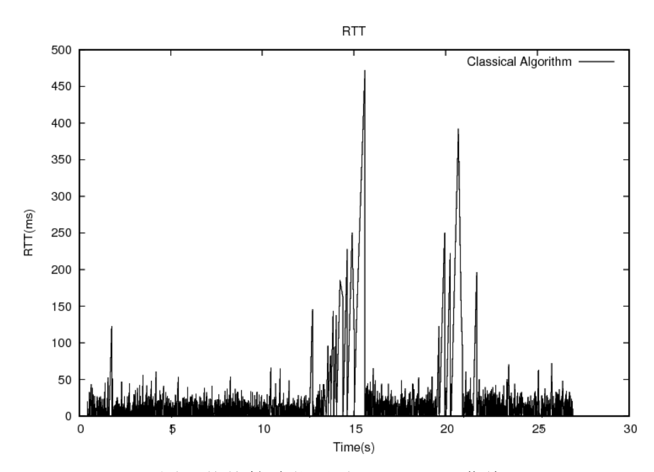


图 4 传统算法往返时延（RTT）曲线

Fig. 4　Traditional algorithm round-trip time (RTT) curve

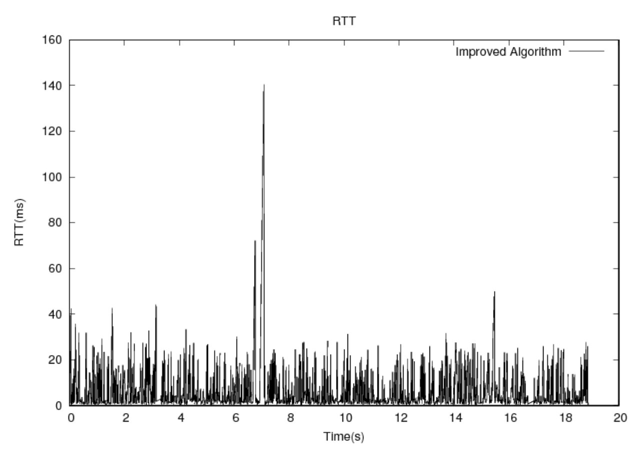


图 5 改进算法往返时延（RTT）曲线

Fig. 5　The round trip time (RTT) curve of the improved algorithm.

图 2 展示了传统慢启动算法下拥塞窗口的变化趋势，在此条件下分别实验了初始拥塞窗口为 3 和 10 的两种情况，从图 2 可以清楚地看到，当 initcwnd=10 时拥塞窗口的增长速度相比 initcwnd=3 时更快，带宽利用率也相对有一定的提高，但它们仍都存在一定的问题，就是拥塞窗口在 TCP 连接启动后期，拥塞窗口增长都过于迅速，很容易就发生了丢包重传的事件，进而导致拥塞窗口急速下降，严重影响 TCP 数据传输的速率，性能很不稳定。而改进的慢启动算法，在启动阶段可以看到，拥塞窗口的增长较为平稳谨慎（图 3），能够很好地跨过慢启动阈值进而进入拥塞避免阶段，很少出现丢包重传的事件，而且可以有效地探测当前的可用带宽，性能较为稳定高效。同样，通过两种算法在传输文件时的往返时延 RTT 曲线图（图 4、5）也能发现改进算法相较于传统算法往返时延较低，而且时延抖动也相对减小，传输性能上有了很大的提高。

另外，本文又在相同的模拟网络环境下利用 FTP 应用传输了大小为 18 0MB 的文件，在文件进行传输的同时，通过使用 WireShark 网络封包分析软件采集了网络中的实时吞吐流量数据，实验的实施同样在两种不同的算法环境下进行，最后绘制成图，实验结果如图 6~ 9 所示。

对比图 6、7 与图 8、9 可以清楚地看到，改进算法在传输文件时网络吞吐量一直保持在一个较高的水平上并保持相对稳定，丢包率也相对较小，而使用传统的慢启动算法的情况下，虽然传输文件时网络吞吐量也相对较高，但却十分不稳定，丢包率相对较高，导致了 TCP 连接的整体吞吐量有所下降。所以，相比之下，改进的慢启动算法具有更好的拥塞控制性能，提高了文件的传输速度。

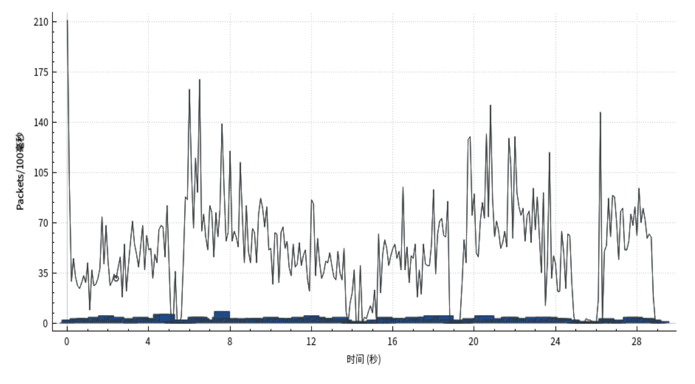


图 6 传统算法下的网络流量变化

Fig. 6　The variation of network traffic under traditional algorithm

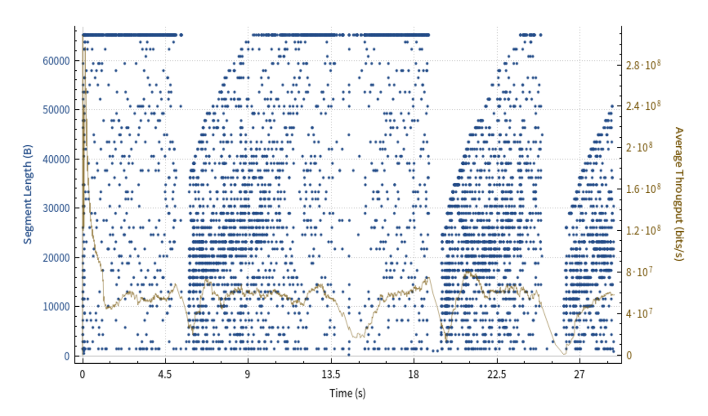


图 7 传统算法下的网络吞吐量

Fig. 7　Network throughput under traditional algorithm

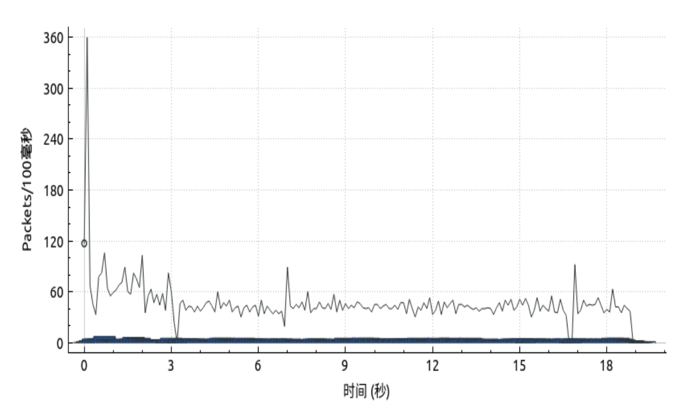


图 8 改进算法下的网络流量变化

Fig. 8　The variation of network traffic under improved algorithm

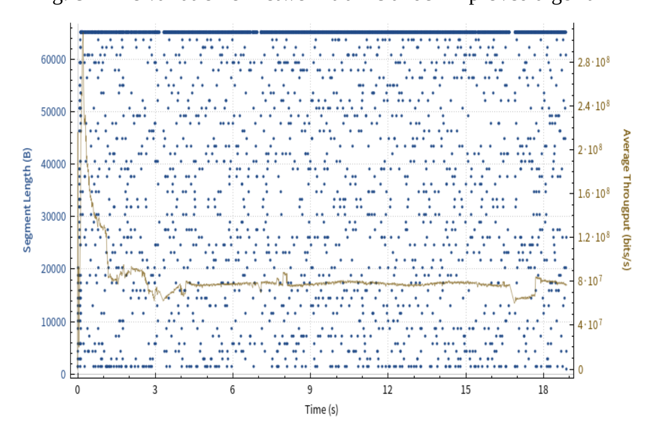


图 9 改进算法下的网络吞吐量

Fig. 9　Network throughput under improved algorithm

3.2　实际网络实验部分

前面已经在模拟的网络环境下验证了改进慢启动算法的性能，下面将在实际的网络环境中进一步测试。本次实验是在校园内的 CMCC-EDU（移动校园无线网）网络环境下进行的，其底层的通信协议是 WLAN，MAC 最大重传次数为 7，无线网的网络最大带宽为 54 Mbps，RTT 平均在 50 ms 左右，本文在同一个 AP（接入点）下分别部署了一台 FTP 服务器和一台 FTP 客户端，FTP 服务器运行在 windows7 操作系统上，FTP 客户端运行在 Linux 系统上，测试的项目是大文件传输速度的测试，分别采用传统的慢启动算法（initcwnd=3）和改进的慢启动算法传输大小为 200 MB、500 MB、1.5 GB 大小的文件，其中使用改进算法的客户端只作为数据的发送方。本文分别使用两种算法各传输 10 次文件，并记录每次传输文件所花费的时间，通过比较使用不同算法情况下文件传输的平均速率来验证改进算法的实际性能。图 10 即为本文在实验测试中获取的数据结果。可以从柱状图中清楚地看到使用改进的慢启动算法在传输 200 MB、500 MB、1.5 GB 大小的文件时速度均比使用传统的慢启动算法时有所提高，传输速率大约提高了 15%左右，可见对可用带宽的使用更加有效。

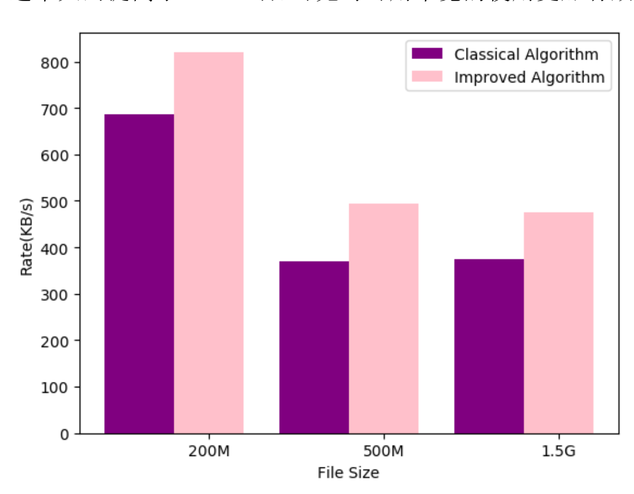


图 10 两种不同算法下文件的传输速率

Fig. 10　Transmission rate of files under two different algorithms

4　结束语

本文讨论了 TCP 慢启动算法的作用和特点，同时全面地分析了传统慢启动算法在高速网络中所存在的缺点和不足，对此，本文结合大文件快速传输以及高速网络环境这两大背景，对传统慢启动算法进行了相应的改进。为了让 TCP 连接在传输大文件时能始终保持一个较高且稳定的吞吐量，并且能适当地提高网络可用带宽的利用率，把研究点主要集中在了如何准确地探测网络可用带宽和网络实时的拥塞状况、如何最大限度地减少数据分组的丢失重传及适当地增加拥塞窗口的大小上。当然，在高速网络中，如果要充分利用网络带宽是不太可能的，因为这需要一个不现实的分组丢失率（2×10^{-10}）[16,17]。改进的慢启动算法对拥塞窗口的增加相对传统慢启动算法来说比较谨慎，它通过尽量减少数据分组的丢失重传和增大初始拥塞窗口来提高文件的传输速度，同时在改进算法中相对弱化了突发 RTT 过大或过小时对拥塞窗口增长速度的影响，减小了窗口的抖动和不稳定性。在模拟的网络实验以及在现实的网络环境中得到的实验结果，表明改进的慢启动算法在性能上取得了一定的提高，大文件的传输速度也有所加快。下一步，将继续通过优化慢启动算法中的拥塞窗口的增长机制及慢启动阈值来进一步提高 TCP 协议的传输效率，使得 TCP 协议在大文件的传输上更加快速有效。

参考文献：

[1] Feldmann A, Rexford J, Caceres R.Efficient policies for carrying Web traffic over flow-switched networks [J]. IEEE/ACM Trans on Networking, 1998, 6(6): 673-685.

[2] Lu Xiao, Zhang Ke, Foh C H, et al. SSthreshless start: a sender-side tcp intelligence for long fat network [J]. Computer Science-Networking and Internet Architecture, 2014, arXiv: 1401. 7146.

[3] Sikdar B, Kalyanaraman S, Vastola K S. Analytic models for the latency and steady-state throughput of TCP tahoe, Reno, and SACK [J]. IEEE/ACM Trans on Networking, 2003, 11(6): 959-971.

[4] Ha S, Rhee I, Xu Lisong. CUBIC: A new TCP-friendly high-speed TCP variant [J]. SIGOPS Operating Systems Review, 2008, 42(5): 64-74.

[5] 王国栋，任勇毛，李俊. TCP 改进协议在高速长距离网络中的性能研究[J].通信学报，2014,35(4):81-90.（Wang Guodong, Ren Yongmao, Li Jun. Performance evaluation of TCP congestion control algorithms in fast long distance network [J]. Journal on Communications, 2014,35(4):81-90.）

[6] Matta I，Guo L, The war between mice and elephants [C] //Proc of the 9th International Conference on Network Protocols.2001:180.

[7] Allman M, Floyd S C. RFC 3390, Partridge. increasing TCP's initial window [S]. 2002.

[8] Dukkipati N, Refice T, Cheng Y, *et al.* An argument for increasing TCP's initial congestion window [J]. ACM SIGCOMM Computer Communication Review, 2010, 40(3): 26-33.

[9] Floyd S. RFC 3742, Limited slow-start for TCP with large congestion windows [S]. 2004.

[10] Choi W, Seckhon R, Seok W. Statistical transmission control based on confidence interval in high bandwidth network [C] Proc of the 2nd International Conference on Internet of Things, Data and Cloud Computing. New York: ACM Press, 2017:1-4.

[11] Wang H, Kang G S, Xin H, *et al.* A simple refinement of slow-start of TCP congestion control[C]//Proc of the 5th IEEE Symposium on.Computers and Communications.2000: 98-105.

[12] Brakmo L S, O'malley S W, Peterson L L. TCP Vegas: new techniques for congestion detection and avoidance [J]. ACM SIGCOMM Computer Communication Review, 1994, 24(4): 24-35.

[13] 屠昊.无线网络中 TCP 拥塞控制机制的研究与实现 [D].南京：东南大学, 2006. (Tu Hao. Research and implementation of TCP congestion control mechanism in wireless network [D].Nanjing: Southeast University, 2006.)

[14] Akamai. The state of the Internet [R/OL]. (2015-01-08). https://www.akamai.com/us/en/about/news/press/2015-press/akamai-releases-third-quarter-2014-state-of-the-internet-report.jsp.

[15] Casetti C, Gerla M, Mascolo S, *et al.* TCP westwood: bandwidth estimation for enhanced transport over wireless links [C]// Proc of the 7th Annual International Conference on Mobile Computing and Networking. New York: ACM Press, 2001: 287-297.

[16] 周冬平，赵奎. 基于RTT的TCP拥塞控制慢启动改进算法 [J]. 计算机与现代化，2015(12): 25-30. (Zhou Dongping, Zhao Kui. An RTT-based TCP congestion control slow-start algorithm [J]. Computer and Modernization, 2015 (12): 25-30.)

[17] 苗广. 基于 Linux 的 FAST TCP 拥塞控制算法优化 [D]. 南京：南京大学，2017. (Miao Guang. Optimization of FAST TCP congestion

control algorithm based on Linux [D]. Nanjing：Nanjing University, 2017.)

[18] Floyd S, Gurtov A. RFC 3649, HighSpeed TCP for large congestion windows [S]. 2003.

[19] 曹涛涛. 拥塞控制算法的性能评估及公平性分析 [D]. 南京:南京大学,2017. (Cao Taotao. Performance evaluation and fairness analysis of congestion control algorithm [D]. Nanjing:Nanjing University, 2017.)